CONQUER: A Tool for NL-based Query Refinement & Contextualizing Code Search Results

Manuel Roldan-Vega, Greg Mallet, Emily Hill, Jerry Alan Fails
Department of Computer Science
Montclair State University
Montclair, NJ, USA
{roldanvegam1, malletg1, hillem, failsj}@mail.montclair.edu

Abstract—**Identifying relevant code to perform maintenance or reuse tasks is becoming increasingly difficult. Software systems continue to grow and evolve, and developers often find themselves searching within thousands to even millions of lines of code to identify code relevant to a particular maintenance task. Automated solutions are vital to help developers become more efficient at locating code to be modified when performing maintenance tasks. In order to address this need and help developers reduce the time spent finding and searching for relevant code, we have built an Eclipse-plugin, CONQUER, that helps developers identify relevant results by providing critical insight and context of how query words are used in the code. CONQUER leverages advanced natural language (NL) information in the source code to group, sort and display the results in a meaningful way. In addition, CONQUER analyzes the frequency and co-occurrence of words in the method result set to provide alternative phrases that can help further refine the query. This rich contextual hierarchy helps the developer to quickly determine if the query is correct and hone in on the relevant results. The NL-based organization of results reduces the number of relevance judgments the developers need to make, and thus can reduce the overall time for a maintenance task.**

Index Terms—**feature location, source code search, software maintenance**

I. Introduction

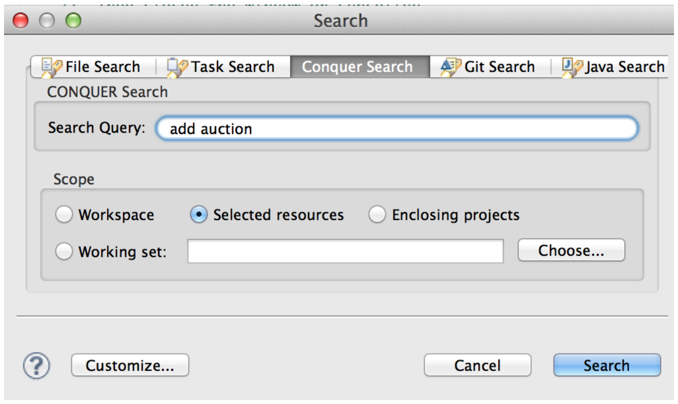
When performing software maintenance or reuse tasks, developers must first identify the relevant code fragments to be modified or reused. In fact, locating the code to be modified is often more time consuming than the maintenance task itself. As software systems continually grow in size and complexity, it becomes more difficult for developers to identify code relevant to a particular task within millions of lines of code. Traditional search tools that use natural language (NL) queries display search results as a simple list [1], [2]. These tools typically do not offer feedback, context, or insight for the developers to understand the results and determine if these are relevant to their query, nor whether the query was successful or not. The developer cannot easily determine if the query is accurate, overly general (and needs to be more specific), or if the query is completely inaccurate. In addition, few of these search tools provide alternative words or phrases that can help the developer refine or reformulate the query. To satisfy this need, we developed CONQUER, an Eclipse-plugin that provides NL context to the developers so that they can identify relevant code or reformulate queries more efficiently and effectively. CONQUER builds on previous research on natural language phrasal representations [3] and source code context [4] and enhances this prior work by analyzing additional contextual information, grouping the results by actions and themes, and suggesting alternative query words. These elements are all integrated into the plugin and made available to the developer for query reformulation and refinement.

II. Background and Related Work

In order to provide developers with useful context on how the query words are used in the source code, it is necessary to identify linguistic factors such as the action and theme of a method signature. The action represents the verb or main activity of the method, whereas the theme is the direct object of that action. The SWUM (Software Word Usage Mode) [3] for Java allows us to capture these linguistic relationships in code. For example, given the signature `addToList(Item)`, SWUM would identify the action as “add” and the theme as “item”. In the cases where there are multiple actions or themes, SWUM also analyzes the head distance to determine the most relevant action or theme. This linguistic information has been used in prior work to improve source code search [3], but has never before been used to group and arrange the query results.

Although the search mechanism is the same as proposed in prior work [3]—which integrates location, semantic role, head distance, and usage information to calculate the relevance score—this query mechanism and results view were not integrated into the Eclipse interface and thus were not usable by the average developer. In contrast, the current approach not only integrates the query mechanism into the Eclipse search dialogue box and displays the results in the view pane; it also provides additional contextual feedback for determining whether the results meet the search needs of the developers. We go beyond the simple hierarchy of phrases used in prior work [4] by organizing the search results by actions and themes. In addition, we automatically recommend alternate query words to help refine overly general or inaccurate queries.

CONQUER differs from previous search tools [1], [2], [5] in that it not only displays the relevant results in a list, but allows the developer to quickly skim a list of actions and themes in the query. In the future, the results could also be presented in the context of a call graph [7], [8], [9]. This work



Search Query: add auction

Scope: Workspace / [✓ Selected resources] / Enclosing projects / Working set: Choose...

Figure 1. CONQUER is an Eclipse plug-in that adds a Conquer Search tab to the general Eclipse Search dialog box.

could be enhanced in future by including semantically related words [11], [10].

III. Tool Overview

The driving insight for our tool is that the same query might be used to search for multiple information needs. Our challenge is in designing a results view that meets the needs of developers searching with queries that are ideal, overly general, partially correct, or completely incorrect. Thus, we designed the CONQUER results view to address 3 key challenges:

1) Quickly determine if results are relevant (i.e., did the query work?).
2) Find alternative query words to refine overly general queries or substitute words in a partially correct query.
3) Find relevant results quickly.

To meet these challenges, we implemented CONQUER as an Eclipse-plugin that allows developers to search any Java software project. The search is integrated into Eclipse's general search dialog box with a CONQUER tab (see Figure 1). After the developer enters a natural language query (i.e., a series of keywords), the plug-in initiates the search mechanism [3].

The CONQUER tool gathers the results obtained from the search and groups them by three categories: action, theme or relevance score. These results are displayed in the view pane in three individual sections in a tree structure based on the action, theme or relevance score. The method signature of each result is displayed as an Eclipse Java element, allowing the developer to click on it to directly navigate to the source code for that method (see Figure 4). While CONQUER can be used to search for methods or fields, for this first phase we have limited the scope to present only methods, leaving fields to future work.

A. *Prevalence of Query Words*

The organization of the CONQUER results view provides the developer with quick feedback on the relative frequency of query words in the result set (see Figure 2). This information allows the developer to quickly determine how well the results match their information need (as expressed by the query), and judge which words should be removed from inaccurate queries.

For example, consider the query "mute music" searching for the "mute" feature in music management software, where the concept of 'mute' is more important to the feature than 'music.' If 'music' is much more prevalent in the source code than 'mute', it will fill the search results with irrelevant entries, causing more work for the developer. The frequencies next to each query word approximate each word's power to discriminate between relevant and irrelevant results. For instance, if 'music' occurs 100 times in the result set and 'mute' just 4 times, the developer will know to revise the query to focus on 'mute', since 'music' is likely to occur with irrelevant results. This will also help the developer focus on the concept of 'mute' in the other parts of the results view.

B. *Suggested Alternative Query Words*

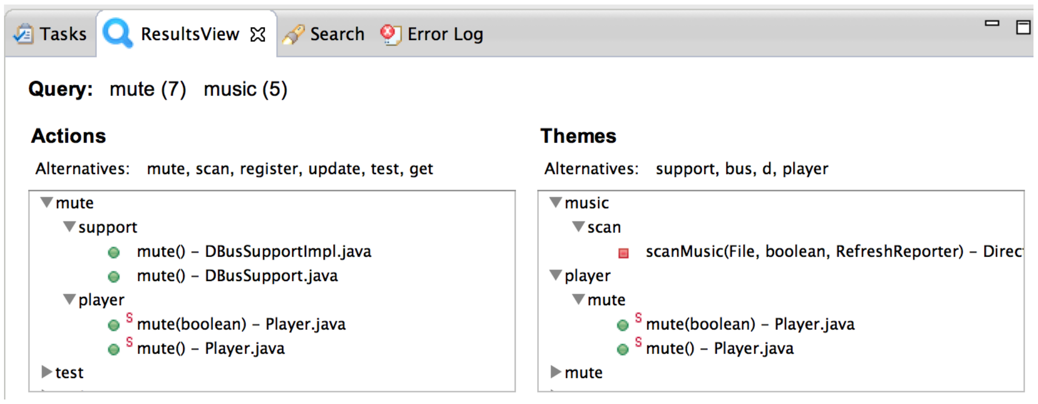
Between the action and theme labels and the tree of action/theme results, we display alternative query words to help reformulate poor queries. As shown in Figure 3, these words can help the developer refine or reformulate the query in cases when the original query is overly general or inaccurate.

For example, consider searching for the shuffling global playlist feature in a music player. A developer might first enter the query "shuffle global playlist", where the ideal query for this particular search task is "shuffle queue". CONQUER will calculate the frequencies of the other action and theme words that co-occur with shuffle and global, and suggest the most highly co-occurring words as alternative query words. As shown in Figure 3, 'queue' would be suggested as an alternative word, helping the developer refine the query to "shuffle global queue", which is much closer to the ideal query.

C. *Results Grouped by Action or Theme*

When a query is not ideal, there still may be relevant results buried in the bottom of the result list with a lower score. We introduce two hierarchies of results, organized by action and theme, to allow more opportunities for a variety of search results to be displayed in the initial result view and facilitate hierarchical navigation of the search results. These two hierarchies are designed to allow the developer to locate relevant results quickly, even for less than ideal queries.

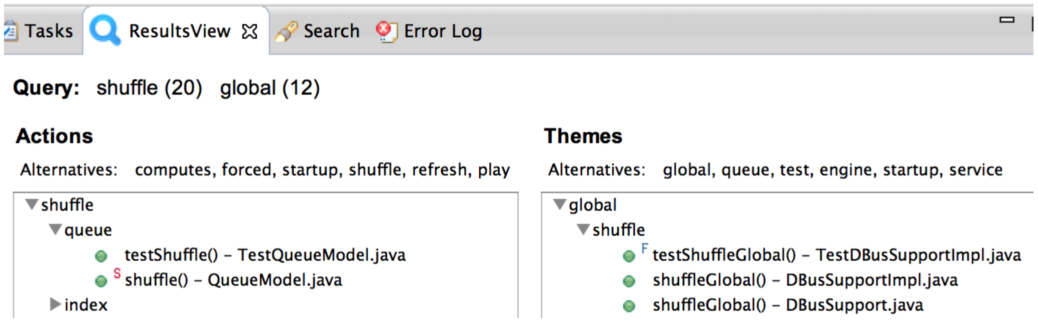
The action section, depicted in Figure 2, displays the search results in a tree structure grouped by the action. All the results that have the same action are grouped into one category. For example, methods saveFile(File) and saveText() would belong to the same action category, "save". Each category has subcategories grouped by theme. In this example, "file" and "text" would both be subcategories. CONQUER calculates the maximum theme score for each subcategory and sorts the categories based on it. CONQUER initially displays a list of all the actions (categories), which can be expanded to see the different themes (subcategories) that appear with that specific action. The developer has the ability to expand



```
Query:  mute (7)   music (5)

Actions                                                   Themes
Alternatives:  mute, scan, register, update, test, get    Alternatives:  support, bus, d, player
▼mute                                                     ▼music
  ▼support                                                  ▼scan
      mute() – DBusSupportImpl.java                             scanMusic(File, boolean, RefreshReporter) – Direc
      mute() – DBusSupport.java                           ▼player
  ▼player                                                   ▼mute
     S mute(boolean) – Player.java                             S mute(boolean) – Player.java
     S mute() – Player.java                                    S mute() – Player.java
▶test                                                     ▶mute
```

Figure 2. Results grouped by action (left) and grouped by theme (right), displayed in a tree structure.



```
Query:  shuffle (20)   global (12)

Actions                                                              Themes
Alternatives:  computes, forced, startup, shuffle, refresh, play     Alternatives:  global, queue, test, engine, startup, service
▼shuffle                                                             ▼global
  ▼queue                                                               ▼shuffle
      testShuffle() – TestQueueModel.java                                 F testShuffleGlobal() – TestDBusSupportImpl.java
     S shuffle() – QueueModel.java                                          shuffleGlobal() – DBusSupportImpl.java
  ▶index                                                                    shuffleGlobal() – DBusSupport.java
```

Figure 3. Alternative query words are shown above the action and theme sections.

each subcategory further and see the method signature of each result, as displayed in Figure 2.

Similar to the action groups, the theme view displays the search results, but the categories are formed by the themes and the subcategories by the actions (see Figure 2). In this view, saveFile(File) and openFile(File) would belong to the same category, "file". Under the "file" category CONQUER displays subcategories "save" and "open". CONQUER calculates the maximum action score for each of the themes and sorts the theme categories based on this score.

D. Results Grouped by Relevance Score

Unlike the tree views of the action and theme section, the bottom of the result view simply displays the results in a list sorted by overall score [3]. This section is most useful when the query is on target and the relevant results are highly ranked.

The result phrase list view is closer to a more traditional search results view, simply listing the methods and files where matches occur. However, we go beyond a simple list view by providing a short natural language phrase that describes each method, to make the results easier to quickly skim without requiring the cognitive overhead of mentally parsing the syntax of method signatures and file names. This facilitates rapidly scrolling through the results, to see what types of words and phrases describe the methods appearing in the result set. In the event of an ideal query, this section may contain the relevant results at the top of the list.

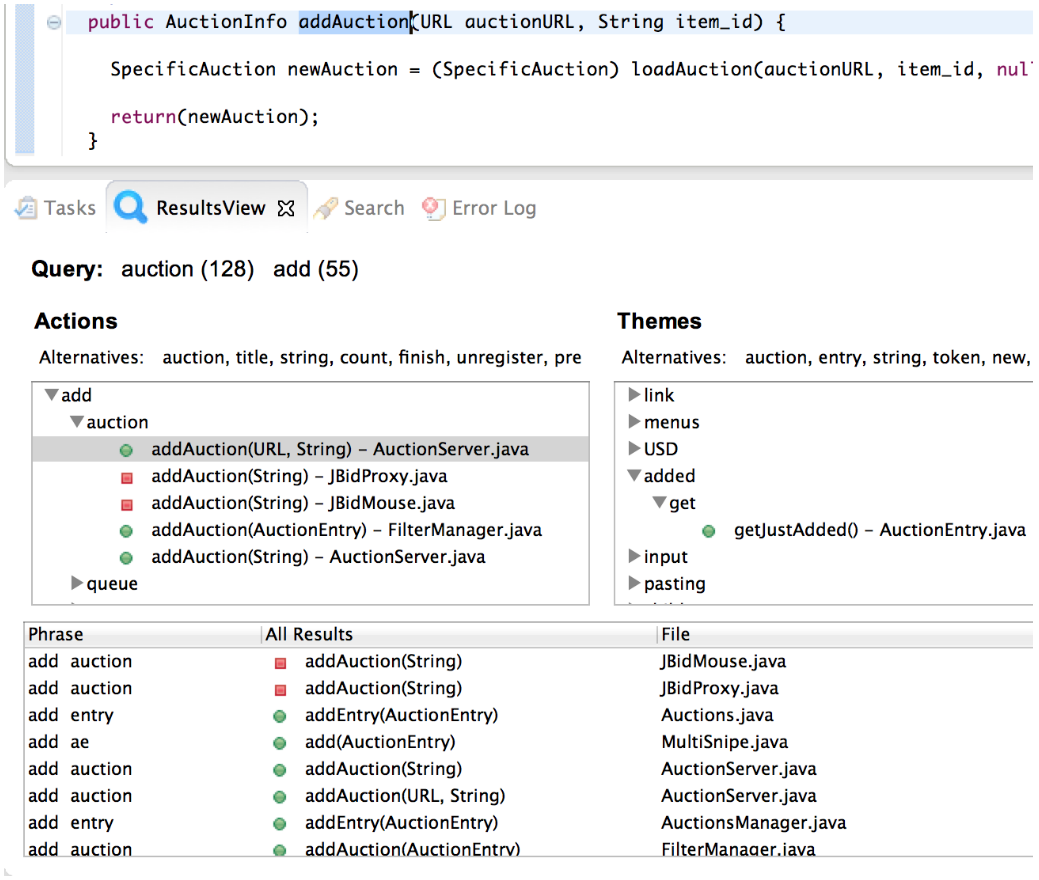
IV. Initial User Feedback

We asked 13 Java developers with industry experience to compare CONQUER (CONQUER) with Eclipse's built-in File Search (ECLIPSE). In analyzing the results, we observed that there are certain situations that lend themselves to each search mechanism. When a developer has an idea of the appropriate identifier names to search for, they want to perform strict matching of identifiers or keywords with an ECLIPSE-like search. In contrast, when a developer is not familiar with a codebase or its naming conventions, and has no insight into what identifiers would be relevant, they need the support of a natural language search provided by CONQUER. Providing a flexible interface for either scenario will further enable the developer to use a single search interface for all their search needs. Future work will investigate how to integrate that customizability in an intuitive way, without using too much screen real estate.

In general, participants appreciated the alternative query words suggested. In fact, some participants requested suggesting synonyms as well co-occurring words. In future, we would like to explore using synonyms as well co-occurring words. In future, we would like to explore using synonyms [11], [10] to enhance the suggested alternative query words.

V. Conclusion

In this paper, we present CONQUER, a tool for NL-based query refinement and contextualizing source code search re-



```
public AuctionInfo addAuction(URL auctionURL, String item_id) {

  SpecificAuction newAuction = (SpecificAuction) loadAuction(auctionURL, item_id, nul

  return(newAuction);
}
```

Query: auction (128) add (55)

Actions

Alternatives: auction, title, string, count, finish, unregister, pre

```
▼add
  ▼auction
      ● addAuction(URL, String) – AuctionServer.java
      🟥 addAuction(String) – JBidProxy.java
      🟥 addAuction(String) – JBidMouse.java
      ● addAuction(AuctionEntry) – FilterManager.java
      ● addAuction(String) – AuctionServer.java
  ▶queue
```

Themes

Alternatives: auction, entry, string, token, new,

```
▶link
▶menus
▶USD
▼added
  ▼get
      ● getJustAdded() – AuctionEntry.java
▶input
▶pasting
```

Phrase		All Results	File
add auction	🟥	addAuction(String)	JBidMouse.java
add auction	🟥	addAuction(String)	JBidProxy.java
add entry	●	addEntry(AuctionEntry)	Auctions.java
add ae	●	add(AuctionEntry)	MultiSnipe.java
add auction	●	addAuction(String)	AuctionServer.java
add auction	●	addAuction(URL, String)	AuctionServer.java
add entry	●	addEntry(AuctionEntry)	AuctionsManager.java
add auction	●	addAuction(AuctionEntry)	FilterManager.java

Figure 4. The CONQUER results view displays 3 sections: grouped by action, grouped by theme, and all results sorted by relevance score.

sults. It allows developers to quickly understand and determine if the query they use to search the source code returns relevant results, and if not, help identify related words to reformulate the query. These insights into the search results can reduce the overall time developers spend in maintenance tasks by helping them locate relevant code more quickly. In future work, we anticipate integrating these techniques into a Visual Studio search framework such as Sando [5].

CONQUER can be downloaded from http://lee.cs.montclair.edu/~hillem/CONQUER/.

REFERENCES

[1] D. Poshyvanyk, M. Petrenko, A. Marcus, X. Xie, and D. Liu, "Source code exploration with Google," in *ICSM '06: Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM'06)*, 2006, pp. 334–338.

[2] T. Savage, M. Revelle, and D. Poshyvanyk, "Flat3: feature location and textual tracing tool," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 255–258.

[3] E. Hill, L. Pollock, and K. Vijay-Shanker, "Improving source code search with natural language phrasal representations of method signatures," in *ASE '11: Proceedings of the 26th IEEE International Conference on Automated Software Engineering, short paper*. Washington, DC, USA: IEEE Computer Society, November 2011.

[4] ——, "Automatically capturing source code context of NL-queries for software maintenance and reuse," in *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, 2009.

[5] D. Shepherd, K. Damevski, B. Ropski, and T. Fritz, "Sando: an extensible local code search framework," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 15:1–15:2.

[6] E. Hill, L. Pollock, and K. Vijay-Shanker, "Exploring the neighborhood with Dora to expedite software maintenance," in *ASE '07: Proceedings of the 22nd IEEE International Conference on Automated Software Engineering (ASE'07)*. Washington, DC, USA: IEEE Computer Society, November 2007, pp. 14–23.

[7] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu, "Portfolio: a search engine for finding functions and their usages," in *Proceeding of the 33rd international conference on Software engineering*, ser. ICSE '11. New York, NY, USA: ACM, 2011, pp. 1043–1045.

[8] G. Scanniello and A. Marcus, "Clustering support for static concept location in source code," in *2011 IEEE 19th International Conference on Program Comprehension (ICPC)*, june 2011, pp. 1 –10.

[9] D. Shepherd, Z. P. Fry, E. Hill, L. Pollock, and K. Vijay-Shanker, "Using natural language program analysis to locate and understand action-oriented concerns," in *AOSD '07: Proceedings of the 6th International Conference on Aspect-Oriented Software Development*, 2007.

[10] J. Yang and L. Tan, "Inferring semantically related words from software context," in *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, 2012, pp. 161–170.

[11] M. J. Howard, S. Gupta, L. Pollock, and K. Vijay-Shanker, "Automatically mining software-based, semantically-similar words from comment-code mappings," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. Piscataway, NJ, USA: IEEE Press, 2013, pp. 377–386. [Online]. Available: http://dl.acm.org/citation.cfm?id=2487085.2487155